**Singapore Management University**
**Institutional Knowledge at Singapore Management University**

Research Collection School Of Information Systems | School of Information Systems

4-2008

# Validating multi-column schema matchings by type

Bing Tian DAI
*Singapore Management University*, btdai@smu.edu.sg

Nick KOUDAS

Divesh SRIVASTAVA

Anthony K.H. TUNG

Suresh VENKATASUBRAMANIAN

**DOI:** https://doi.org/10.1109/ICDE.2008.4497420

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the Databases and Information Systems Commons

## Citation

DAI, Bing Tian; KOUDAS, Nick; SRIVASTAVA, Divesh; TUNG, Anthony K.H.; and VENKATASUBRAMANIAN, Suresh. Validating multi-column schema matchings by type. (2008). *IEEE 24th International Conference on Data Engineering 2008 ICDE: Cancun, Mexico, April 7-12: Proceedings*. 120-129. Research Collection School Of Information Systems.
**Available at:** https://ink.library.smu.edu.sg/sis_research/4167

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

# Validating Multi-column Schema Matchings by Type

Bing Tian Dai*, Nick Koudas#, Divesh Srivastava†, Anthony K. H. Tung*, Suresh Venkatasubramanian‡

*National University of Singapore
Singapore 117590, Republic of Singapore
daibingt@comp.nus.edu.sg
atung@comp.nus.edu.sg

#University of Toronto
Toronto, ON M5S 2E4, Canada
koudas@cs.toronto.edu

†AT&T Labs–Research
Florham Park, NJ 07932, USA
divesh@research.att.com

‡University of Utah
Salt Lake City, UT 84112, USA
suresh@cs.utah.edu

***Abstract*— Validation of multi-column schema matchings is essential for successful database integration. This task is especially difficult when the databases to be integrated contain little overlapping data, as is often the case in practice (e.g., customer bases of different companies). Based on the intuition that values present in different columns related by a schema matching will have similar "semantic type", and that this can be captured using distributions over values ("statistical types"), we develop a method for validating 1-1 and compositional schema matchings.**

**Our technique is based on three key technical ideas. First, we propose a generic measure for comparing two columns matched by a schema matching, based on a notion of information-theoretic discrepancy that generalizes the standard geometric discrepancy; this provides the basis for 1:1 matching. Second, we present an algorithm for "splitting" the string values in a column to identify substrings that are likely to match with the values in another column; this enables (multi-column) 1:m schema matching. Third, our technique provides an invalidation certificate if it fails to validate a schema matching. We complement our conceptual and algorithmic contributions with an experimental study that demonstrates the effectiveness and efficiency of our technique on a variety of database schemas and data sets.**

## I. Introduction

Schema matching is an important problem in the realm of database integration. The basic operator is called *Match* [1], and associates schema elements from one database with schema elements from another database, as a prelude to schema mapping and database integration. Early approaches to schema matching were based on identification using only schema labels. This was followed by techniques that took into account the actual data in the columns being matched. When schema matching is performed between databases that are likely to contain the same data (e.g., the catalogs of different online book sellers), the matching can check for occurrences of the same values across the databases. To allow for the possibility that schema matching is performed between databases that contain different data (as is more commonly the case), there has been work on describing data in a column via aggregate distributions and using inferences on these distributions to identify matches. Schema matching has numerous applications, and is an area of extensive study; surveys by Rahm and Bernstein [1] and Doan and Halevy [2] cover the major issues and much of the relevant literature.

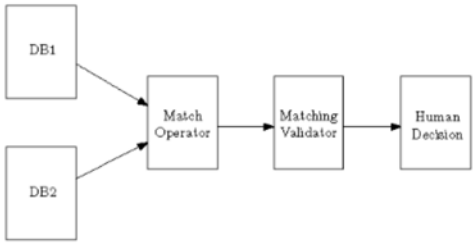



Fig. 1. The flow of Schema Matching

Essential to the success of schema matching (and hence of database integration) is a *validator* that, given a candidate match, declares it to be valid or not. As shown in Figure 1, the match operator takes two databases, and gives a set of candidate matches. The "match validator" then plays a role of declaring if a given candidate match is valid. According to the result of the match validator, humans can make a decision to keep or discard a match. If a schema matching is declared to be invalid, a good validator should provide a "certificate of invalidation" to facilitate human decision. It is harder to provide a "proof" of a valid matching, since such a proof merely implies the lack of contradicting evidence. Every technique for discovering schema matchings in the literature

TABLE I

americaDB AND chinaDB: SCHEMA AND SAMPLE DATA

| americaDB | | | | |
|---|---|---|---|---|
| propId | propTitle | piFN | piLN | coPiName |
| 05CN0734 | who likes xml | john | pardon | wei li |
| 06CN0732 | who hates xml | johnny | walker | lei cai |

| chinaDB | | |
|---|---|---|
| proposalInfo | piName | coPiName |
| 1136-AA-654 05 i like xml too | li mingfei | johnson bush |
| 1136-B-45 06 the future of xml | yu kai | pitts daniel |

implicitly includes a validator, possibly in conjunction with a search strategy for identifying candidate matchings. In this paper, we focus on the validation aspect of schema matchings and hence our technique can be combined with any search strategy that helps identify candidate matchings.

### A. String Values and Statistical Types

Surprisingly, despite the prevalence of string values in databases (e.g., person names, company names, building locations, email addresses), and the vast amount of work in schema matching, little attention has been paid to validating matchings of string-valued schema elements. Exceptions include systems like CUPID [3] and the work by Embley et al. [4] that consider multi-column matchings of the form "concatenate FN and LN" (i.e., string concatenation), and the recent work by Warren and Tompa [5] who study the discovery and validation of multi-column substring matchings of the form "concatenate the first 7 characters of LN and the first character of FN". A limitation of these previous techniques is that their validator can be used only in database integration scenarios where matching schema elements have the *same values*, but not in the more common case where matching schema elements have different string values (e.g., when two merging companies integrate their customer databases).

A validator for such data needs to determine if the two columns contain data of the same "semantic type". Examples of semantic types include customer IDs, titles, names, prices, *etc*. This is a classic problem in information retrieval, where one goal is to group documents by topic; in this case, the semantic type of a document is its topic (or set of topics), and documents with similar semantic type are grouped together.

The central idea of this paper is based on extensive research in information retrieval suggesting that semantic type can be modeled by "statistical type", defined by distributions of $q$-grams of strings drawn from the text.

Comparing two string-valued columns then becomes the problem of estimating a distance between the distributions corresponding to their (possibly many) types. Building on these intuitions, this paper addresses the challenging problem of validating multi-column schema matchings by examining the statistical types defined by columns of data.

### B. An Illustrative Example

Validating by type forces us to revisit some of our basic ideas about matching schemas. Consider two databases maintaining information about collaborative multi-national research proposals, `americaDB` and `chinaDB`: `americaDB` tracks all research proposals where the primary investigator is based in the US, and `chinaDB` tracks all research proposals where the primary investigator is based in China. Their schemas and some sample data are given in Table I.

Note that all the data in these databases are represented as strings. This is fairly prevalent in real databases, due to the flexibility afforded by the use of strings. When exploring and validating schema matchings between these two databases, possibly to create an integrated target database `chinamericaDB` with the schema `(propTitle, americaPerson, chinaPerson)`, there are several issues that need to be addressed.

*No guarantee of common data:* There is no guarantee that the principal investigators or co-principal investigators in one database will be present in the other database. Thus validating a candidate schema matching cannot be based on entire string values. However, there is the possibility of using $q$-grams to aid in the schema matching. Note (in the data of Table I) that 2-grams like "jo" and "on" seem to occur more commonly in American names while 2-grams like "ai" and "ei" seem to occur more commonly in Chinese names. By looking at 2-grams, one could invalidate a candidate matching between `americaDB.coPiName` and `chinaDB.coPiName`.

*Simple and composite matchings:* This example illustrates the presence of both simple and composite matchings. For example, the matching between `americaDB.coPiName` and `chinaDB .piName` is a simple 1-1 matching, and should be validated. Since the integrated database `chinamericaDB` uses single columns to represent both American names and Chinese names, the validator would need to be able to validate a multi-column composite matching between `concat (americaDB .piLN, americaDB.piFN)` and `chinaDB.coPiName`, but invalidate a candidate multi-column composite matching between `concat (americaDB.piFN, americaDB .coPiName)` and `chinaDB.coPiName`.

*Substring matchings:* Since `propId` is not part of the target schema, to populate attributes like `propTitle` in the integrated database `chinamericaDB`, one would need to validate a matching between `americaDB.propTitle` and `substring (chinaDB.proposalInfo)`. Note that a candidate matching between `americaDB.propTitle` and `chinaDB.proposalInfo` should not be validated.

### C. Integratability

How then do we determine if the match between two string-valued columns is valid? To do this, we introduce the notion of *integratability*. As we discussed earlier, we can represent each string by a distribution over $q$-grams, and define the statistical types of a column as a collection of distributions, one for each string. Then, we expect that if two such collections represent similar statistical types, then *any reasonable clustering of the distributions will populate clusters with data from the two collections in roughly the same ratios.*

An example of this idea is shown in Figure 2. In both panels, the data is depicted geometrically, with crosses representing data from one column, and squares from the other. In the left hand panel, the two columns exhibit similar type distributions, and a reasonable clustering puts the same relative fractions of data from the two columns in each cluster. Note that unless the two columns have exactly the same number of items, these ratios need not be balanced (i.e., 50-50). All we expect is that the proportion is relatively constant across clusters. In the right hand panel, the data from the two columns separate out into different clusters, and thus there are clusters with widely varying fractions of crosses and squares.

This notion can be made precise in an information-theoretic manner; in a later section we will see that this idea translates into computing a certain kind of conditional entropy. A key aspect of this notion is that it allows us to exploit *context*; since the data from both columns are clustered together, the process discovers commonalities between the data that may not be apparent if the two columns were clustered independently. It also allows us to quantify the degree of integratability in a purely generic manner, with no recourse to metric spaces and embeddings of data.

### D. Validating Substring Matches

String data may have complex internal structure, and it is often the case that a schema match arises by splitting a string value into substrings, and matching the substrings with elements from another schema. Formally, this means that we need a substring extraction oracle that, when applied to a column of string-valued data, returns a new column on which we can run an integratability check. However, the only information we have regarding the location of these substrings is the column we wish to integrate with.

Our idea is thus as follows: rather than examining strings in the column for obvious internal delimiters that might mark the boundaries of a relevant substring, we use the target column to find such substrings. We use type information extracted from this column to segment each string and return the most promising candidate substring. We do this by evaluating the likelihood of individual $q$-grams of the string appearing in strings from the target column. If we find a set of contiguous $q$-grams that have an *unusually high* likelihood, we have found our desired candidate substring.

### E. Validating Composite Matchings

Validating schema matchings by type allows us to capture composite associations that would be difficult via value-based methods. Certain composite associations reduce to a simple match problem, that is, determining whether the concatenation of data from two columns matches with data from another column reduces to validating a simple match.

Consider however, the problem of determining whether one set of columns *jointly* matches another set of columns. In our example from Section I-B, this happens when matching names; since labeling of names as *first* and *last* may be inconsistent, especially when dealing with names from Asian countries. In this case, any approach based on concatenation of column values will struggle to find a match, because there may be no consistent way of ordering data to find a match. Using our distributional approach however, we are indifferent to the issue of finding orderings. We can concatenate the columns and do a type-based validation for the super-columns; the $q$-gram extraction approach allows us to identify common regions without considering their location.

### F. Invalidation Certificate

Recall that if a schema matching is declared to be invalid, a good validator should provide a "certificate of invalidation". The final component of our work is a certificate of invalidation, presented as a "bar code", which will come from the clustering itself. A low integratability score indicates that the data from the two columns separate into different clusters in our soft clustering, which can be presented to the user.

### G. Our Contributions

In this work, we will present a method for validation of multi-column schema matchings based on using statistical type structure in the data to identify semantic types. Our specific contributions are as follows:

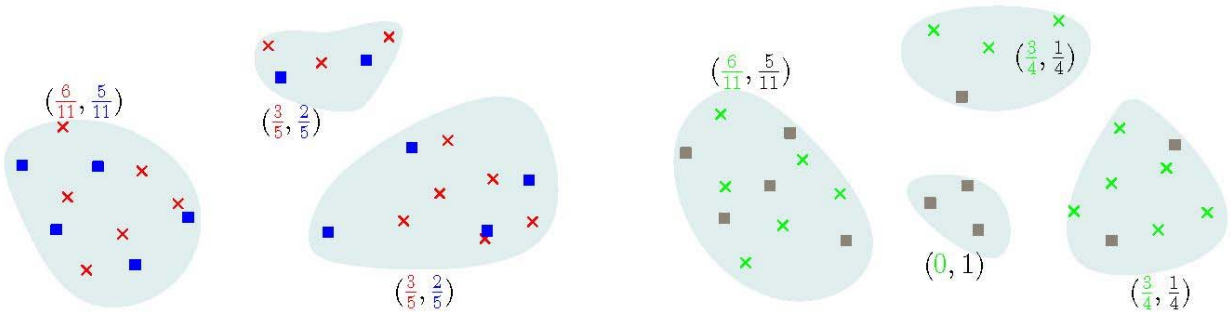
- A generic type-based *integratability measure* based on information-theoretic considerations, and an algorithm to compute this measure (Section IV).
- An algorithm for validating substring matches that uses a likelihood-based substring extraction method (Section V).
- Procedures for validating single column and multi-column schema matchings based on the tools developed above (Section VI).
- An invalidation certificate if the algorithm fails to validate a schema matching (Section VII).
- An experimental study that demonstrates the effectiveness of our technique on a variety of database schemas and data sets (Section IX).

We discuss related work in Section II. Technical details of our implementation are presented in Section VIII.

## II. Related Work

Schema matching and schema mapping have been studied extensively. The survey by Rahm and Bernstein [1] lays out a general ontology of approaches, classified by the level of granularity (schema-level, data-level), the kinds of rules generated (1-1, 1-m), and other factors. A later survey by Doan and Halevy [2] covers more recent work, as well as related research in AI and machine learning.

To the best of our knowledge, there is no prior work that formulates the idea of matching schema based on inferred semantic types from string data. Related work includes [6] and [4] (which constructs types from domain knowledge rather than from the data). There are however a number of works that use statistical and/or machine learning methods to learn properties of an attribute from data and examples. Some notable examples are SEMINT [7], and work by Berlin and Motro [8], [9]. Kang and Naughton [10] use the mutual information of




(a) Each cluster contains roughly the same proportion of the two data sets.

(b) The cluster vary greatly in the relative ratios of the two data sets.

Fig. 2. Computing integratability between two sets of data. In each case, a candidate clustering is given. The two sets are marked with crosses and squares.

two columns as a measure of how likely they are to be matched to each other. In their scheme, the data is effectively treated as categorical. Other related works in this area include the work of He, Chang and Han [11] on schema matching for the deep web and work on using distributional signatures for value mapping by Kang *et al* [12] and Naumann *et al* [13]. For related work in the AI community, we refer the reader to the survey by Doan and Halevy [2].

In this paper, we validate schema matchings, rather than discovering them. Notable methods for *discovering* schema matchings (and mappings) include Clio [14], which is a comprehensive schema mapping tool. SPIDER [15], built on top of Clio, is a data-driven debugger for schema mappings that uses the idea of *routes* to express mappings between schemas. In terms of finding substring matches, a closely related work is by Warren and Tompa [5], who study the discovery of multicolumn schema mappings. In one sense, their problem is harder than ours, because they discover potential schema maps, while we validate such maps. However, their method assumes explicit value mappings, and in that sense inhabits a more restricted space than our approach.

Another related work is the paper by Dai *et al* [16], which uses information-theoretic concepts to estimate the diversity or *heterogeneity* of a column of data for data cleaning purposes. Although mutual information makes an appearance in that work as well, the measure describes a property of a *single* data set, and behaves very differently to the notion of integratability (defined on a *pair* of data sets) discussed here.

## III. Definitions

Let $X$ be a random variable taking on values $x_1, \ldots x_n$. The *entropy* $H(X)$ is the entropy of the distribution $p_i = p(x_i) = p(X = x_i)$, and is given by $H(X) = -\sum p_i \log p_i$. The *conditional entropy* of a random variable $X$ given another random variable $Y$ is denoted by $H(X|Y)$, and is computed by the formula $H(X|Y) = H(X,Y) - H(Y) = -\sum_{x_i,y_j} p(x_i, y_j) \log p(x_i|y_j)$. The *mutual information* of random variables $X$ and $Y$ is defined as $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$. Intuitively, the more correlated $X$ and $Y$ are, the higher their mutual information is; independent random variables have zero mutual information.

We will represent statistical types via soft clusterings of data. In a *soft clustering*, each item is associated with potentially many types, and the weights of these associations sum to one. Formally, we will denote the association weights by the conditional probabilities $p(t|x)$, which represents the weight with which the point $x$ is associated with type $t$. Note that $\sum_t p(t|x) = 1$, and that a hard clustering corresponds to setting exactly one of the $p(t|x)$ values to one. Associated with the clustering are the *prior* weights $p(x)$.

A $q$-gram of a string $s$ is any substring of $s$ of length exactly $q$. For example, the set of 1-grams of a string is the multi-set of all the letters in the string.

## IV. A Measure of Integratability

Let $C_1$ and $C_2$ be two columns of data that we wish to integrate based on their types. Consider any soft clustering of the union of $C_1 \cup C_2$, with associated conditional probabilities $p(t|x)$ for each $x \in C_1 \cup C_2$ with each $t \in T$, the set of types. Considering the union of the columns to have unit probability mass, we will denote by $p(C_i)$ the quantity $|C_i|/(|C_1|+|C_2|)$, where $|C_i|$, the *weight* of $C_i$, is equal to its cardinality if all items are considered equally significant. For each column $C_i$ and each type $t$ compute the *type distribution*

$$p(t|C_i) = \sum_{x \in C_i} p(t|x)p(x)/p(C_i)$$

By our informal definition of integratability, two columns have the same type structure if the two type distributions are similar. A standard measure for comparing distributions $p$ and $q$ is the Jensen-Shannon distance [17], defined as

$$\mathrm{JS}(p, q, \alpha, \beta) = \alpha \mathrm{KL}(p, m) + \beta \mathrm{KL}(q, m)$$

where $m = \alpha p + \beta q, 0 < \alpha, \beta < 1, \alpha + \beta = 1$, and $\mathrm{KL}(p, q)$ is the Kullback-Leibler distance $\mathrm{KL}(p, q)$, also known as the *relative entropy* [18].

The distance between the two type distributions is then $\mathrm{JS}(p(t|C_1), p(t|C_2), p(C_1), p(C_2))$. Some algebra reveals this as the mutual information $I(C;T)$, where $C = \{C_1, C_2\}$ is the set of columns.

$I(C;T)$ is larger when the columns are less integratable, and has a maximum value of $\min(H(C), H(T))$. It is more convenient for a measure of integratability to be larger when the two columns are *more* integratable, and be normalized to the range $[0,1]$. Therefore, using the relation $I(C;T) = H(C) - H(C|T)$, we define our measure of integratability as the expression

$$\text{Integratability} = \frac{H(C|T)}{H(C)} \quad (1)$$

which takes values in the range $[0,1]$ and is maximized when the two columns are identical.

Consider the two instances depicted in Figure 2. In both examples, there are 15 crosses and 11 squares, and therefore $H(C) = 0.983$. For the left side, $H(C|T) = 0.98$, and therefore the integratability is 0.998. On the right side, $H(C|T) = 0.795$, and thus the integratability is 0.81, which is less than 0.998, as expected.

*Integratability Versus Separability:* One might design a classifier-based approach to determine the integratability of two data sets. The classifier would receive training data labeled as being from one column or the other, and use this training data to classify the remaining data from the two columns. Intuitively, if the classifier is unable to identify class labels well, the data is integratable. Unfortunately, this approach does not work, and illustrates a key difference between the notion of *separability* inherent in the use of classifiers, and the notion of integratability. The column data is typically represented as points in a high dimensional space. In order for a classifier to be successful, it must be able to find a set of dimensions in which a separating hyperplane (or more general separating structure) can be constructed. In other words, the goal is to find a local dissimilarity; if one exists, the data can be classified into separate groups.

However, the goal of integratability is to capture *global similarities* in the data, aggregated over the entire space. Two columns of data might be similar in all but a few respects, and thus are integratable; however a classifier will hone in on the points of difference and separate the data. Consider the example from Section I-B. Suppose we had two tables from `chinaDB` and wished to validate a proposed match between the `piName` columns in each table, where in one table, the PIs listed were from mainland China and in the other table, the PIs listed were from Hong Kong. Constructing two such columns with 200 names each, the integratability measure gives a score of 0.9968, which means these two columns are perfectly integratable. We then used a Naive Bayes classifier from the Weka toolkit [19] to build a classification of column A and column B by supplying half of the data as training data, labeling each item with the column it came from. We then used the other half data to test the classifier, which correctly classified 178 instances out of 200. In other words, although the integratability of the two columns was high, reflecting the global similarity in the names, the separability (as indicated by the success of the classifier) was also high, showing that local differences could be used to distinguish the data.

## V. Extracting A Match

A measure of integratability allows us to determine whether two columns of data have the same type. A more general kind of schema match occurs when a single column (which we will call the heterogeneous column) contains parts that identify with other columns. We came across one such example from Section I-B, when attempting to populate the attribute `propTitle`. In such cases, one would not expect a simple column-level comparison to detect type similarities; instead, what we need is a method for determining a *substring* of a column that matches types with another column.

As before, let $C_1$ and $C_2$ be the two columns under consideration. Let $C_1$ be the *heterogeneous* column, i.e. the column from which we wish to extract a substring whose type matches that of $C_2$ (which we call the *corpus*). For each string $s \in C_1$, we need to determine a substring $w(s)$ such that the set of strings $\{w(s)|s \in C_1\}$ integrates with $C_2$.

Suppose we have a procedure to compute a candidate $w(s)$. We can then test the efficacy of this procedure by integrating the resulting strings with $C_2$, using the method outlined in Section IV. A good procedure will generate a high integratability score.

One way of generating a substring is by considering absolute positions in a string. For example, the substring extraction described by Warren and Tompa [5] calls for specific offsets into a string to generate the target string. It is possible (although inefficient) to search over all offsets to find the correct substring. However, in general such an approach will fail to capture many kinds of heterogeneous data, whose values may be derived semantically from different sources. This motivates our approach, which is based on the *type* of the homogeneous column in order to extract relevant substrings.

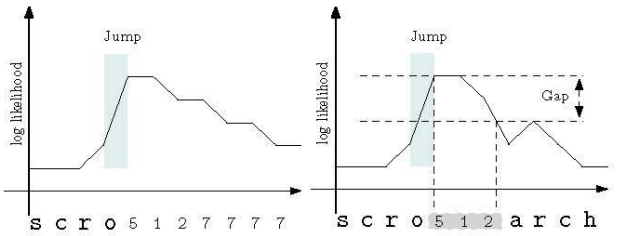
*Finding A Large Bump:* We assume that a string in the column $C_1$ is of the form $awb$, where $w$ is a string whose type matches that of $C_2$ and $a, b$ are arbitrary strings. If this is the case, then we expect that a representation of $w$ in the type space will match closely with a representation of strings in $C_2$, while the representations of $a, b$ will not match as well.

We build a histogram on strings that characterizes the corpus $C_2$; in practice, we fix a parameter $q$ and compute frequency counts for all $q$-grams that occur in any string of $C_2$.

Now for each string $s \in C_1$ we extract its $q$-grams and assign to each a weight that equals the relative frequency of this $q$-gram in the histogram constructed above. Let the relative range of these frequencies (the ratio of the largest to smallest) be denoted by $r$.

Treating these relative frequencies as likelihoods, we expect that if $w$ indeed matches strings in $C_2$, then the likelihood of $q$-grams in $w$ should be significantly higher than that in the remainder of the string. In other words, as we move from left to right along $s$, we should see a jump in likelihood where $w$ starts, and a corresponding dip where it ends.

The magnitude of a jump that constitutes a significant event is a subjective choice, depending on $r$. A good choice for the jump magnitude is $r/2$, for reasons we shall see later.




(a) Jump in likelihood indicates interesting transition (b) *low*, *high* regions indicate interesting substring

Fig. 3. A substring extraction example

Consider a scenario where the corpus consists of different kinds of numeric sequences. A possible likelihood plot for the string `scro5127777` might look like Figure 3(a). Here, the jump (the shaded region) marks the beginning of the relevant substring.

The figure also illustrates how finding a significant transition is not sufficient. In the above example, there is no way to determine which of the strings `512`, `5127`, `51277` etc. are the appropriate strings to return, without making some kind of *ad hoc* choice.

Therefore, rather than defining a transition as a sufficiently large jump from one $q$-gram to the succeeding one, we define a transition as a separation of likelihoods into two sets, *low* and *high*, such that the smallest likelihood in *high* is significantly larger than the highest likelihood in *low*.

If such a separation exists, then it is easy to identify the corresponding substring: we label all $q$-grams as being either *high* or *low*, and take the longest substring consisting of high $q$-grams.

Figure 3(b) depicts how this works, with the string `scro512arch`. Here, we can identify a clear gap of size $r/2$ that separates the $q$-grams into regions. The resulting substring that the algorithm returns is shaded in the figure.

The choice of $r/2$ as the gap width guarantees that we find at most one high substring. If we find no region, we deem the whole string to be high and return it as is.

## VI. (In)Validating Schema Matchings

Having developed tools for comparing and extracting types, we now apply them to different validation scenarios.

*1-1 schema matching:* The first scenario is a match between two columns, under the assumption that data in the first matches the type of data in the second, for example, the matching between `americaDB.coPiName` and `chinaDB.piName`. We compute integratability as described in Section IV. We have observed that integratability values close to 1.0 are reliable indicators of a valid match; in any case, comparing integratability values is a reliable way of deciding which matches are more accurate than others.

*Compositional matchings:* Compositional matchings can be validated by using a simple concatenation strategy as described in Section I; the use of distributions over $q$-grams allows us to use the above method for validation.

*Substring Matches:* Another kind of matching that has no analogue in the value-based world is a substring matching. In this scenario, a schema matcher proposes a match between two columns of a table, but the actual matching types are parts of the values. Again, using the multinational research proposal example, we need to use substring matching to populate attributes like `propTitle` in the integrated database `chinamericaDB` from `americaDB.propTitle` and `substring (chinaDB.proposalInfo)`.

In general, the match might occur between substrings of both columns. We use the procedure of Section V to independently extract substrings from each column using the other column as the corpus, and then check the integratability between the sets of extracted substrings from the two columns.

## VII. An Invalidation Certificate

The type-based validation methods described above all end with an integratability calculation. The score is larger when the two matching schema elements are believed to be more similar in type. If they are believed to be dissimilar in type, besides the integratability score, we should also provide a "certificate of invalidation".

A hard clustering is easy to represent by listing the partitions. A soft clustering is harder to visualize as points spread their mass across many clusters. Moreover, as pointed out in [16], even the notion of a cluster is not well-defined. Two distinct clusters are not really distinct if the membership probabilities $p(t|x)$ are identical for both of them.

We will visualize integratability using the idea of a "barcode"[1]. Consider an image with one row for each item in either of the two columns, grouped so that rows for items in $C_1$ appear above rows for items in $C_2$. Each column of this image corresponds to a cluster $t$, and has a variable width proportional to $p(t)$. The content of a column is the value $p(t|x)$, represented in logscale with darker shades signifying higher probabilities.

Since a soft clustering might spread mass between essentially identical clusters, we need to group the columns for such clusters together, to make the representation meaningful. Two clusters are similar if their cluster membership vectors $p(t|x)$ are similar. With this observation, we can group clusters together, and reorder the clusters by their groups.

*An Example:* We illustrate the idea of a bar code with three simple examples, drawn from name databases (see Section IX for the details on the data). One of the integratability tests described in Section I was a test to see whether two columns of names corresponded to the same nationality. Suppose we were to compare lists of American and German names. Figure 4 illustrates what the barcode looks like. As described above, rows are indexed by the data, and columns are indexed by clusters. Each entry of the barcode represents a particular cluster membership probability $p(t|x)$, with a darker

[1]The idea of using a bitmap to visualize a soft clustering was used in [16]. However in that work, the only goal of the visualization was to check that a data set was clustered.

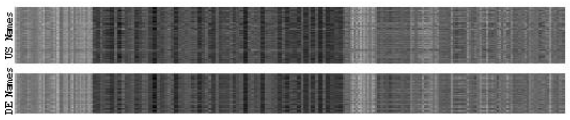



Fig. 4. Integratability of US and DE names $= 0.99$

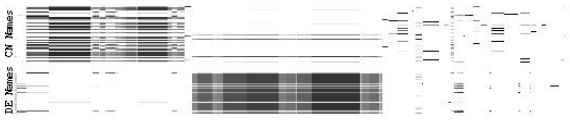



Fig. 5. Integratability of CN and DE names $= 0.24$

color representing a higher value (in logscale). All entries of the first data source appear above those of the second.

The integratability of these two sources is very high; notice that in the bar code, the two horizontal strips look almost identical. This means that each cluster contains the same mass from both data sets, implying high integratability. Note that given the Germanic roots of many American names, this result is not particularly surprising.

At the other extreme, consider doing the same with Chinese and German names. The integratability score is as low as $0.24$, and predicts that the data sets should separate out into separate clusters. As Figure 5 shows, this is exactly the case. The top and bottom strips are almost complementary to each other, indicating that most clusters contain mostly one kind or the other, and rarely both. The shades are also deep, indicating not only a difference, but a strong difference.

An intermediate example can be seen by integrating American and Russian names, which bear some, but not significant resemblance. The integratability score here is $0.61$, and as Figure 6 shows, the top and bottom strips are not as distinct as in the Chinese-German case, and neither are they as identical as in the American-German case. Remember that the intensity of the shading indicates probability mass, and in this case the shading is weaker, indicating that even when there appears to be differences, it is not as significant.

## VIII. Methodology

The previous section discussed our main type-checking tools and how they may be applied for schema match validation. In this section we now discuss the details of our implementation.

### A. Preparing The Data

We treat all data as strings, regardless of content. We represent strings using $q$-gram distributions. For each string, we extract all its $q$-grams, and construct a histogram, with the entry for each $q$-gram recording its frequency. Using a large value of $q$ seemingly allows us to capture more sophisticated patterns in the data; however, since the number of potential $q$-grams (the number of "dimensions") grows exponentially with $q$, the effect of these (fewer number of) $q$-grams is muted. Algorithms that manipulate these representations will also perform much worse as $q$ increases. We have found in our experiments that setting $q \leq 2$ provides a good balance between the efficiency of the computation and the accuracy of the results. Thus, we use $q$-grams of length one and two.

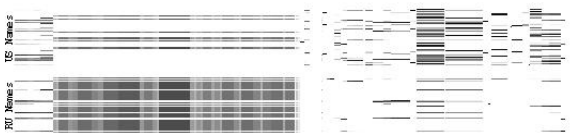



Fig. 6. Integratability of US and RU names $= 0.61$

*Weighting the q-grams:* In information retrieval, where $q$-gram representations are most prevalent, individual terms are weighted based on measures like *term frequency (t.f)* and *inverse document frequency (i.d.f)*. The primary purpose of information retrieval systems is search, and so rare terms (with large i.d.f) are important to reduce the size of search results.

In our setting, weighting schemes like i.d.f are not appropriate. Our goal is to determine whether two columns of data can be integrated; therefore, $q$-grams that *distinguish* one set of data from another are more important than $q$-grams that are either rare or frequent.

To clarify this, consider integrating two columns of data with the same number of items. A $q$-gram with no distinguishing power will appear in roughly the same number of items in each column. A $q$-gram with high distinguishing power will occur mostly in one column or the other.

There are different ways of constructing a measure that captures this idea. The approach we will use is as follows. For a $q$-gram $q$, we compute the number of strings it appears in within each column; call these frequencies $f_1, f_2$. Taking the difference and normalizing by $n$, the number of items in each column, we get the quantity $p' = (f_1 - f_2)/n$ that ranges between $-1$ and $1$ (if the cardinalities are different, we compute $f_1/n_1 - f_2/n_2$ instead). It will be convenient to work with the range $[0, 1]$, so we shift and scale $p'$, obtaining $p = (1 + p')/2$.

This new variable has a range of $[0, 1]$, and is $0$ or $1$ when the $q$-gram appears exclusively in every string in one or the other columns. It takes the value $0.5$ when the $q$-gram appears equally often in both. We desire a function that is symmetric around $0.5$ and is maximized at the boundaries. Any appropriately chosen convex function suffices; we will use shifted negative entropy $W(p) = 1 - H(p)$, where $H(p) = -p \log p - (1 - p) \log(1 - p)$. Once the frequency counts are weighted using the above method, we normalize the histograms, yielding a distribution for each string.

### B. Computing Integratability

Once we have a collection of distributions, we employ the soft clustering method first developed in [16]. In brief, the algorithm fixes a quality-compression tradeoff parameter $\beta$, augments the data with a background, and then runs the `iib` algorithm developed by Tishby, Pereira and Bialek [20], [21]. The method then removes the background, yielding a soft clustering of the data. Computing integratability is now straightforward; we apply the formula 1 from Section IV.

### C. Substring Extraction

We first compute the relative frequencies (likelihoods) of $q$-grams using the target column. Once this is done, substring extraction from a string $s$ proceeds in two steps. First, we must determine the threshold that separates *high* $q$-grams from *low*, and then we must find the longest substring consisting of *high* $q$-grams alone.

We first sort the $q$-grams of $s$ by their log-likelihood, and compute the gap parameter $g = 0.5(\max - \min)$ where $\max, \min$ are the maximum and minimum log-likelihood values respectively. Then, we find a pair of consecutive $q$-grams in the sorted order such that the difference between their log likelihood is at least $g$. If such a pair exists, then the lower likelihood marks the threshold between *low* and *high*. We can now tag each position of $s$ with a mark that indicates whether the $q$-gram starting at this position is *high* or *low*, and find the longest *high* substring. If no such pair exists, the entire string is returned.

## IX. Experiments

We now present detailed experiments of our integratability measure and its application on schema matching validation. We will first describe the platform and the data sets, followed by the experiments on our integratability measure. Lastly, we will look at the effectiveness of this measure for validating schema matchings.

### A. Platform and Datasets

The machine we run our experiments on has a Intel(R) Pentium(R) 4 2.40GHz CPU and 2GB of RAM. It runs Fedora Core 5 with kernel version 2.6 and gcc version 4.1.1. In our experiments, we have utilized three datasets:

**`Contact Data`**: This is the data set used in [16]. This data set contains five columns identifying information with different semantic type: email, id, ip, circuit-id and phone number. Each individual column has a single type, i.e., the email column is a set of email addresses.

**`Name Data`**: This data is drawn from a rank list at *acm.uva.es*. People are associated with their country. We selected some real names from this list, and grouped them by their regions.

**`Book Data`**: This data set consists of book catalogs with different schema [22]. Example attributes are *title* and *price*.

### B. Validation of Integratability Measure

In this section, we will focus on validating our integratability measure. We will provide examples whose semantic type can be understood intuitively, and show that our measure in fact matches with the intuition. We will first consider the simplest case: both columns are simple (not compositional) fields.

*1) Integratability between two columns:* First, we apply the integratability measure on the `contact` dataset. Results are shown in Table II.

TABLE II
Integratability of Contact Data

| | Column A | | | | |
|---|---|---|---|---|---|
| Column B | Email | ID | ip | circuitID | phone |
| Email | 1.00 | | | | |
| ID | 2.10e-09 | 1.00 | | | |
| ip | 8.07e-10 | 1.17e-3 | 1.00 | | |
| circuitID | 8.23e-10 | 7.66e-9 | 5.55e-8 | 1.00 | |
| phone | 1.48e-09 | 0.222 | 0.152 | 1.23e-5 | 1.00 |

First it is evident that the measure satisfies reflexivity; every column is integratable with itself. The columns are quite distinct from each other, so this is reflected in the low integratability scores for pairs. Notice that ID and phone have a low integratability with each other, but it is not as low as the other pairs. This is likely because many of the IDs are seven digit numbers, similar to phone numbers.

TABLE III
Integratability of Names between Different Groups of People

| Names from Different Region | Integratability with CN Names | Integratability with HK Names |
|---|---|---|
| CN Names | 1.00 | 0.997 |
| HK Names | 0.998 | 1.00 |
| US Names | 0.202 | 0.441 |
| DE Names | 0.212 | 0.480 |
| RU Names | 0.0922 | 0.264 |
| ES Names | 0.175 | 0.341 |

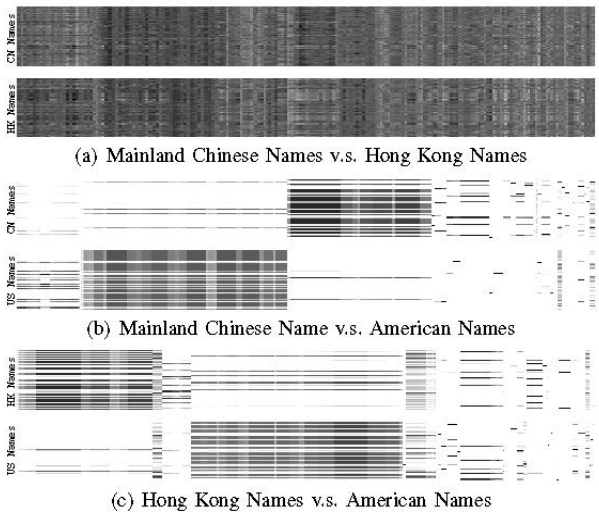
In the `name` dataset, we grouped names from the same region as a single column. For example, mainland Chinese usually use the romanization of their Chinese names as their names in the Latin system. Chinese people from Hong Kong or Singapore will probably use the anglicization of their Chinese names, with another English name as a prefix or suffix, like Simon Cheung.

From Table III, we discovered that although Hong Kong names are very integratable with mainland Chinese names (and vice versa), Western names are more likely to integrate with names from Hong Kong rather than names from mainland China. This matches with our intuition that Hong Kong names are a combination of Chinese names and English names, and it is noteworthy that a purely information-theoretic approach is able to capture this.

Investigating further the properties of this measure, the example shown in Figure 7 is instructive. In this example, we showed the barcodes obtained for the integratability of pairs of name sets between Chinese, Hong Kong and American names. Note that transitivity does not hold; although names from China and America both integrate well with names from Hong Kong, they do not integrate well with each other.

*2) Validation of Substring Extraction:* In this section, we will validate our method of substring extraction by concatenating two columns and testing this *heterogeneous* column against one of the two original columns (the *corpus*).

In the first example, given a heterogeneous column containing concatenations of circuitID, ip address, email address and phone number, e.g., "`agec.840948..ati172.30.`




(a) Mainland Chinese Names v.s. Hong Kong Names

(b) Mainland Chinese Name v.s. American Names

(c) Hong Kong Names v.s. American Names

Fig. 7. Barcodes for 3 Pairs of `name` data

TABLE IV
CN NAMES EXTRACTION FROM A HETEROGENEOUS COLUMN OF CN AND US NAMES

| Concatenated String | Substring Extracted |
|---|---|
| *mingfei li*christopher johnston | mingfei li |
| *guo ce*gregory lee | guo ce |
| *shen yi*thomas grindinger | shen yi̲ |
| *kai yu*daniel pitts | kai yu |
| *wenyuan dai*johny walker | wenyuan daij |
| *xin yangfeng*phoenix gabriel | xin yanfeng |
| *liu su*nathan backman | liu su̲n |

`136.3atzeni@dia.uniroma3.it913 780-3664`", "`agec.681001..ati135.184.182.165hirata@ccrl.sj.nec.com713-805-4233`" and a set of email addresses as the corpus, substrings like "`atzeni@dia.uniroma3.it`", "`hirata@ccrl.sj.nec.com`" are extracted, which shows our scheme correctly identified the email addresses from the heterogeneous column.

In the second, and harder, scenario, we attempt to extract Chinese names from strings concatenated from Chinese names and western names. The result is shown in Table IV, and the right column contains substrings extracted from the left column. The text in *italics* in the left column of Table IV is the original Chinese names which we used to construct the concatenated string. We underline the incorrect portions of the string extracted.

Firstly, notice that the string extracted is almost exactly the same as the concatenated string, and in few incorrect situations, only a few extra characters are extracted. It is also worth noting that the purpose of substring extraction is a subroutine to check integratability, so a perfect extraction procedure, although desirable, is not critical for the entire procedure to work correctly. Indeed, the integratability of the extracted strings with the corpus is 0.92.

TABLE V
INTEGRATABILITY OF SUBSTRINGS BETWEEN TWO VERTICALLY HETEROGENEOUS COLUMNS

| Column A | Column B | Integratability |
|---|---|---|
| CN Name | US Name | 0.241 |
| CN Name-Email | US Name-Email | 0.850 |
| CN Name-ID | ID-US Name | 0.716 |

TABLE VI
PAIR-WISE INTEGRATABILITY OF TABLE `books1.csv` AND TABLE `books2.csv`

| | Title | Author | Listprice | Price |
|---|---|---|---|---|
| Title | 1.00 | 0.431 | 2.94e-12 | 3.14e-14 |
| Author | 0.446 | 1.00 | 5.17e-10 | 1.61e-11 |
| Listprice | 1.45e-12 | 3.25e-10 | 0.956 | 0.272 |
| Price | 3.34e-14 | 3.59e-11 | 0.313 | 0.910 |

*3) Integrating two heterogeneous columns:* An interesting question that we did not fully address was how extraction of substrings will work if both columns under consideration were heterogeneous themselves. For example, we could concatenate our "email" and "id" columns to create a new column "email-id", which consists of strings concatenated from "email" followed by "id". The columns obtained by such concatenation are vertically heterogeneous.

One approach to deal with this case is to run two instances of substring extraction. We first use the first column as a corpus to extract the substrings from the second column. This will give us a set of substrings which are most similar to the first column. We then do the same thing again, but using the second column as corpus to extract substrings from the first column. Lastly, we put the two sets of substrings together and compute their integrability.

Table V shows the integratability between two pairs of columns: CN Name-Email against US Name-Email and CN Name-ID against ID-US Name. The first row, the integratability between CN Name and US Name, provides a reference point for the latter rows.

We see that in both cases, substring extraction from both columns is moderately successful at extracting the relevant common portions from both strings; the integratability scores are much higher than the scores for the base (non-integratable) pair of Chinese and American Names.

### C. Schema Validation

`Book data` contains two tables with the same schema, "books1" and "books2". We list a pair-wise column integratability result in Table VI.

An interesting discovery we make from Table VI is that the fields `Price` and `ListPrice` do not integrate.

Looking closer at some of the sample values for these two entries, the last two digits of `ListPrice` are always of the form `.00` or `.95`, e.g. `14.00, 15.95, 29.95,`

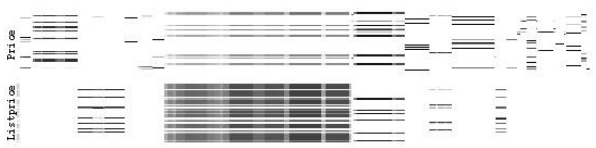



Fig. 8. Barcodes for `books1.csv.Price` vs `books2.csv.Listprice`

TABLE VII
PERFORMANCE: ALL TIME IN SECONDS

| Data Columns | Preparation | #q-grams | Clustering Time per iteration |
|---|---|---|---|
| Title-Author | 0.61 | 722 | 0.143 |
| Publisher-Author | 0.32 | 584 | 0.160 |
| Price-ISBN | 0.08 | 140 | 0.0266 |
| Category-Quantity | 0.19 | 474 | 0.192 |
| Rating-Listprice | 0.05 | 62 | 0.0446 |

`12.00`, whereas the last two digits of `Price` are more varied, e.g. `107.10, 12.89, 15.63, 68.39`. The two attributes are listed differently, and therefore it is correct to treat them as distinct, which our integratability measure allows us to do. However, treating these fields as numeric fields, and using (for example) distributions on the numerical values to distinguish them is unlikely to work since the distribution on prices is very similar to the distribution on list prices. We note that when the `Price` field in one table is integrated with its corresponding field in the other table, we get the high integratability scores that we expect.

Figure 8 illustrates the bar code obtained when attempting to integrate `Price` and `ListPrice`.

### D. Performance

In this section, we evaluate the performance of the algorithm by providing a detailed break down of the time required. The running time is recorded while doing column matching validation for the `book` data, listed in Table VII. We note that in general, our algorithm scales *linearly* with the number of individual matches to be validated; this should be contrasted with the discovery problem, where one has to deal with a combinatorial explosion of candidate matches.

Each experiment is run with 200 sampled rows of data from each column. The main bottleneck is the time taken to perform the soft clustering. This is an iterative algorithm that typically runs for between 10-20 iterations. The field `#q-grams` indicates the dimension of the resulting histograms once data preparation is complete. Note that the histograms are usually extremely sparse.

## X. Conclusions

In this paper, we introduced the idea of validating schema matchings by type, and developed a set of tools for validating different types of schema matchings using type information. Our method uses information-theoretic methods and is thus generic; however, it is also effective at capturing detailed semantic structure in data.

Concrete domain-specific knowledge can help validation of schema matchings greatly, and one interesting line of future work would be to explore ways in which context can be given to our generic method to exploit domain knowledge. One example of this could be the use of more specific priors. Integrating our approach into a larger schema identification system like Clio [14] would also be quite interesting.

## References

[1] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB Journal*, vol. V10, pp. 334–350, 2001.
[2] A. Doan and A. Y. Halevy, "Semantic-integration research in the database community," *AI Mag.*, vol. 26, no. 1, pp. 83–94, 2005.
[3] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic schema matching with cupid," in *Proc. VLDB*, 2001, pp. 49–58.
[4] D. W. Embley, L. Xu, and Y. Ding, "Automatic direct and indirect schema mapping: experiences and lessons learned," *SIGMOD Rec.*, vol. 33, no. 4, pp. 14–19, 2004.
[5] R. H. Warren and F. W. Tompa, "Multi-column substring matching for database schema translation." in *Proc. VLDB*, 2006, pp. 331–342.
[6] A. Doan, P. Domingos, and A. Y. Halevy, "Reconciling schemas of disparate data sources: a machine-learning approach," in *Proc. SIGMOD*, 2001, pp. 509–520.
[7] W.-S. Li and C. Clifton, "Semint: a tool for identifying attribute correspondences in heterogeneous databases using neural networks," *Data Knowl. Eng.*, vol. 33, no. 1, pp. 49–84, 2000.
[8] J. Berlin and A. Motro, "Autoplex: Automated discovery of content for virtual databases," in *Proc. CoopIS*, 2001, pp. 108–122.
[9] J. Berlin and A. Motro, "Database schema matching using machine learning with feature selection," in *Proc. CAiSE*, 2002.
[10] J. Kang and J. F. Naughton, "On schema matching with opaque column names and data values," in *Proc. SIGMOD*, 2003, pp. 205–216.
[11] B. He, K. C.-C. Chang, and J. Han, "Discovering complex matchings across web query interfaces: a correlation mining approach," in *Proc. KDD*, 2004, pp. 148–157.
[12] J. Kang, D. Lee, and P. Mitra, "Identifying Value Mappings for Data Integration: An Unsupervised Approach," in *Proc. WISE*, 2005.
[13] F. Naumann, C.-T. Ho, X. Tian, L. M. Haas, and N. Megiddo, "Attribute classification using feature analysis." in *Proc. ICDE*, 2002, p. 271.
[14] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin, "Data-driven understanding and refinement of schema mappings," in *Proc. SIGMOD*, 2001, pp. 485–496.
[15] L. Chiticariu and W. C. Tan, "Debugging schema mappings with routes." in *Proc. VLDB*, 2006, pp. 79–90.
[16] B. T. Dai, N. Koudas, B. C. Ooi, D. Srivastava, and S. Venkatasubramanian, "Rapid identification of column heterogenity," in *Proc. IEEE Intnl. Conf. Data Mining*, 2006.
[17] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Trans. on Information Theory*, vol. 37, no. 1, pp. 145–151, 1991.
[18] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York, NY, USA: Wiley-Interscience, 1991.
[19] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. Morgan Kaufmann, 2005.
[20] N. Tishby, F. Pereira, and W. Bialek, "The information bottleneck method," in *Proc. 37-th Annual Allerton Conference on Communication, Control and Computing*, 1999, pp. 368–377.
[21] N. Slonim, "The information bottleneck: Theory and applications," Ph.D. dissertation, The Hebrew University, 2003.
[22] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster, "Putting context into schema matching." in *VLDB*, 2006, pp. 307–318.